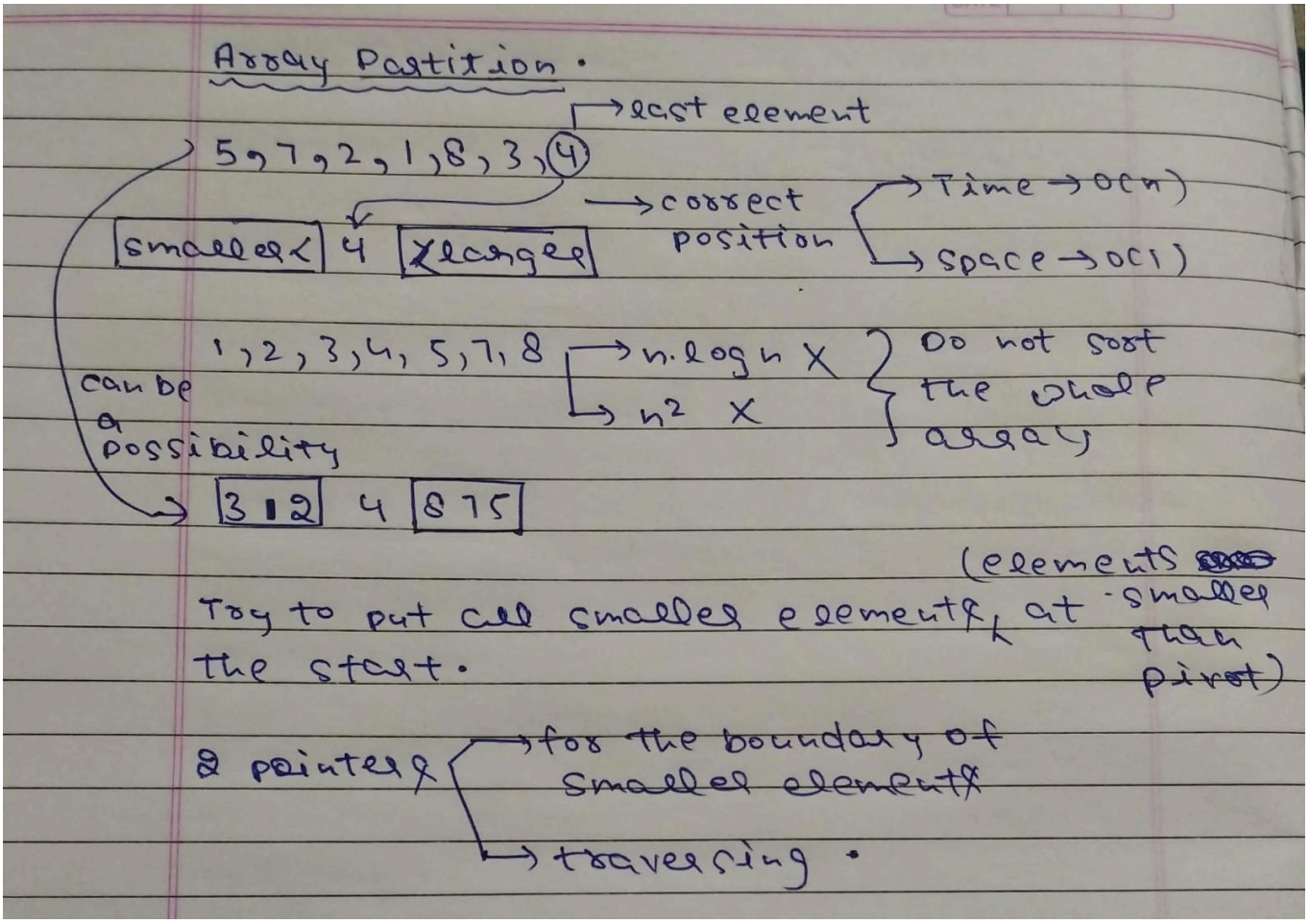
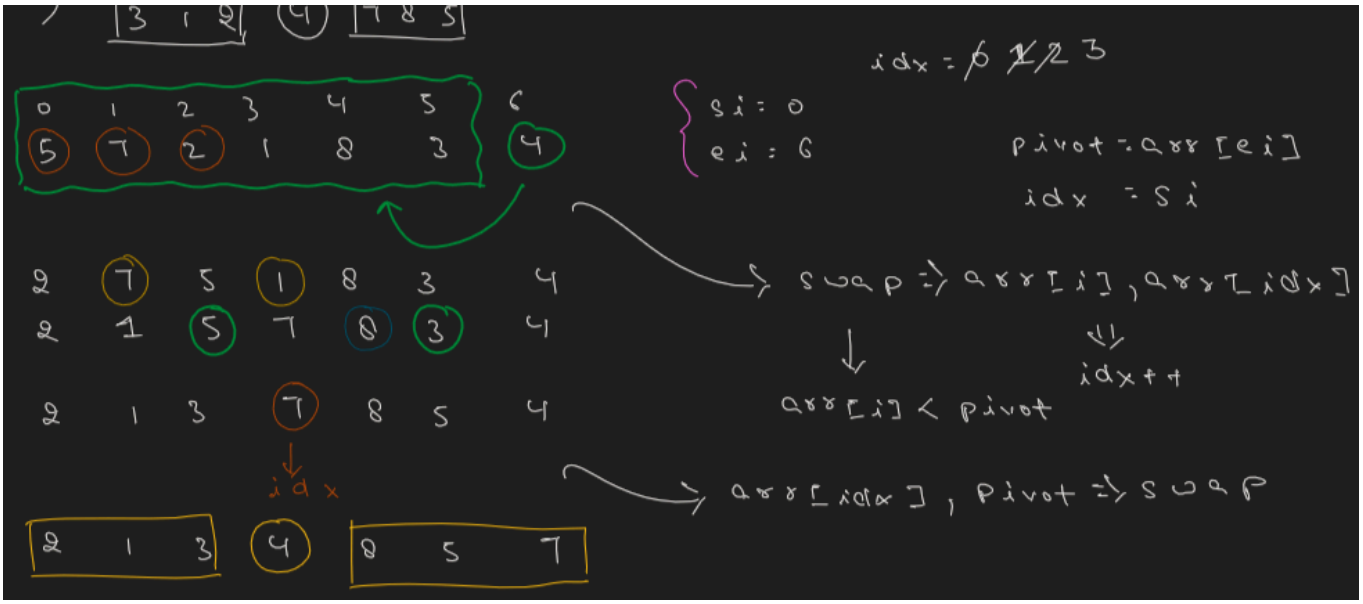
🔀 Partition in Array



The pivot can only be placed at a position from index 0 to n - 2. If it cannot be placed within this range, it means the pivot is already at its correct position.




```java
public class Main {
	public static void main(String[] args) {
		int[] arr = { 5, 7, 2, 1, 8, 3, 4 };
		System.out.println(partition(arr, 0, arr.length - 1));

		for (int i = 0; i < arr.length; i++) {
			System.out.print(arr[i] + " ");
		}
	}
```

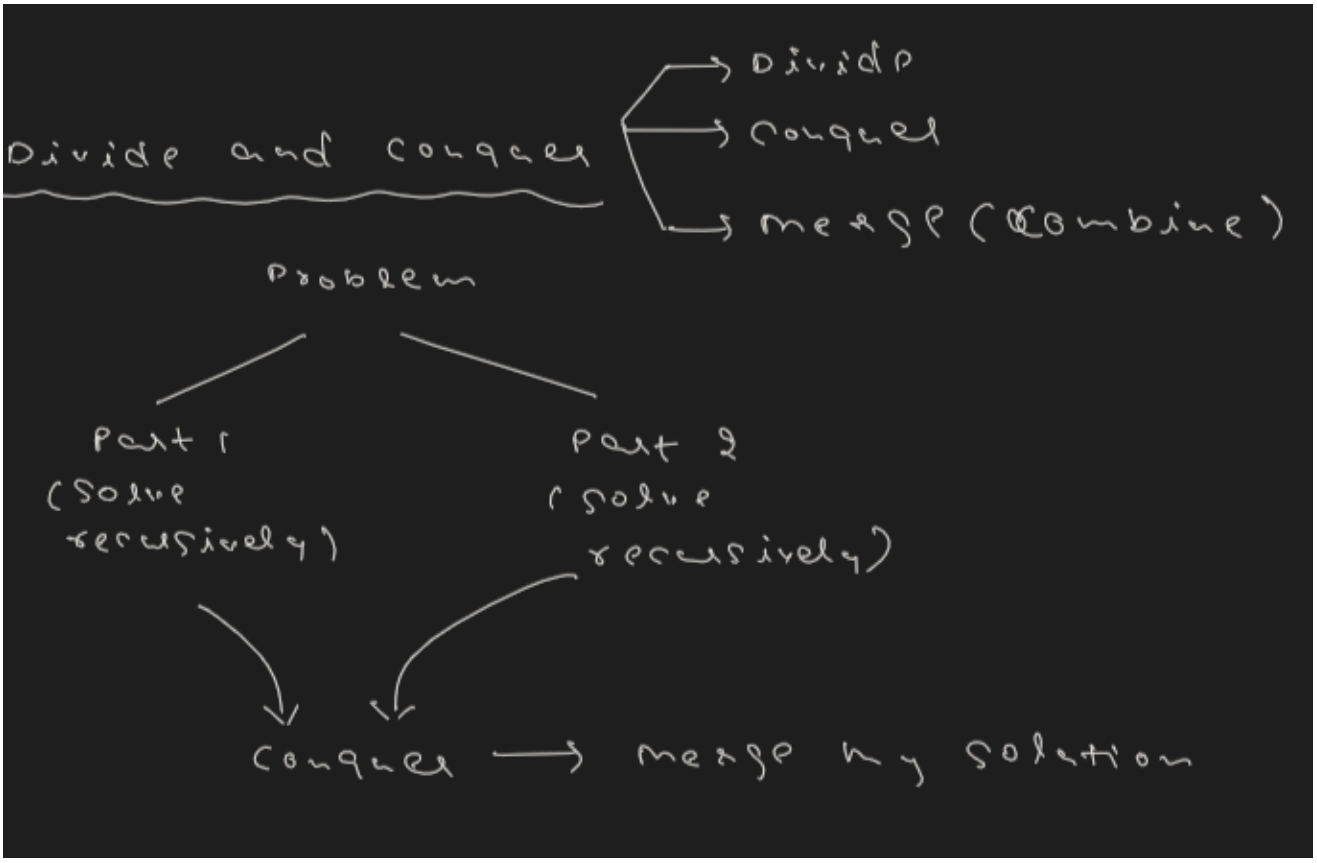
```java
    public static int partition(int[] arr, int si, int ei) {
        int item = arr[ei];
        int idx = si;
        for (int i = si; i < ei; i++) {
            if (arr[i] <= item) {
                int temp = arr[i];
                arr[i] = arr[idx];
                arr[idx] = temp;
                idx++;
            }

        }
        int temp = arr[ei];
        arr[ei] = arr[idx];
        arr[idx] = temp;
        return idx;
    }

}
```

In this problem, we are using the last element as the pivot, then performing the swapping to partition the array.

Now, suppose we want to use any other element as the pivot. In that case, we simply swap that element with the last element, and the rest of the algorithm remains the same.

⚔️ What is Divide and Conquer?

Divide and Conquer is a powerful algorithmic strategy used to solve complex problems by breaking them down into simpler sub-problems, solving each of those independently, and then combining their results to get the final solution.



🧩 How It Works

The strategy follows **three main steps**:

1. **Divide**:
   Break the problem into smaller sub-problems of the same type.
2. **Conquer**:
   Recursively solve each sub-problem. If the sub-problem is small enough, solve it directly.
3. **Combine**:
   Merge the solutions of the sub-problems to form the final result.

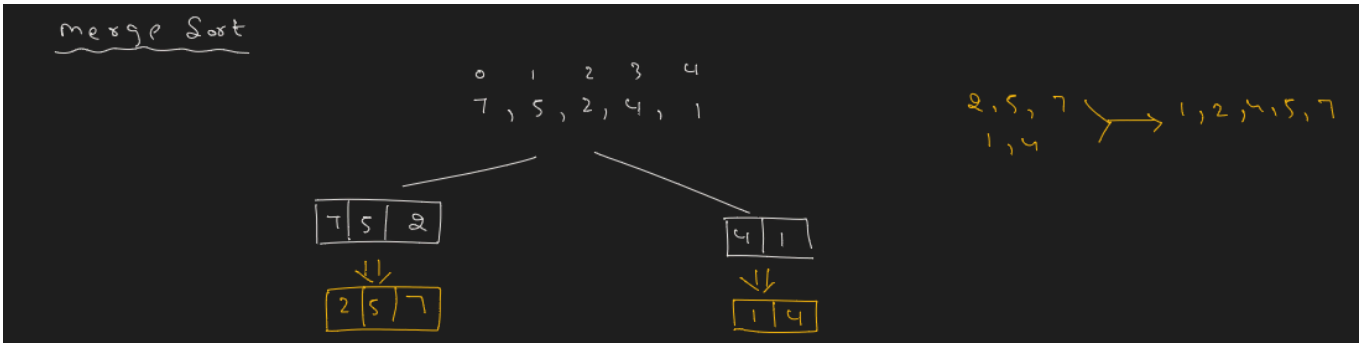
🧵 What is Merge Sort?

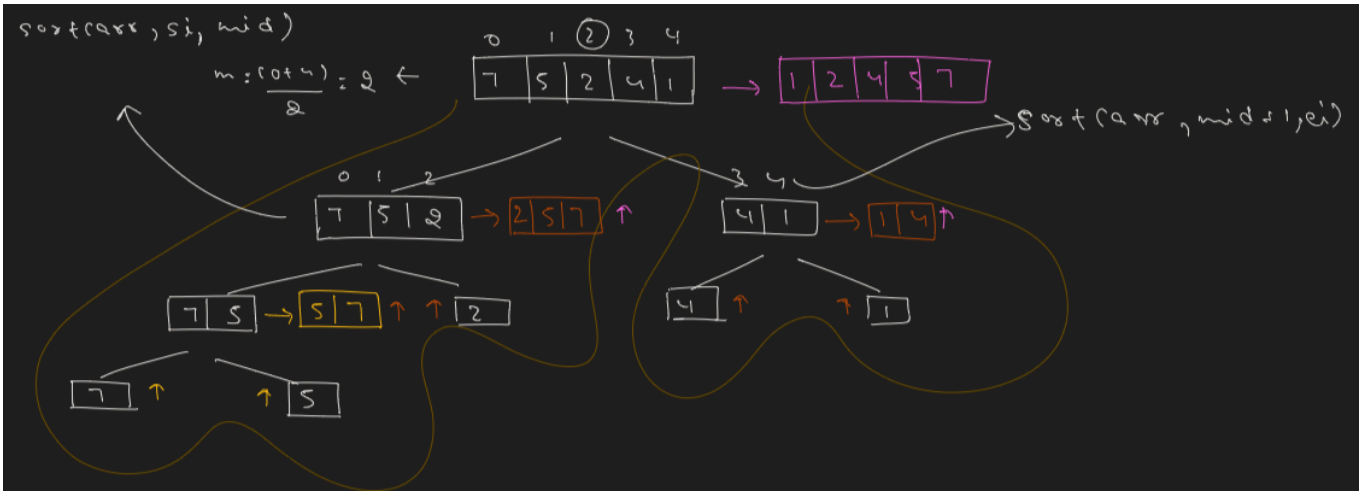
Merge Sort is a classic **Divide and Conquer** algorithm used for **sorting** an array efficiently. It has a guaranteed time complexity of **O(n log n)**, making it very efficient even for large datasets.

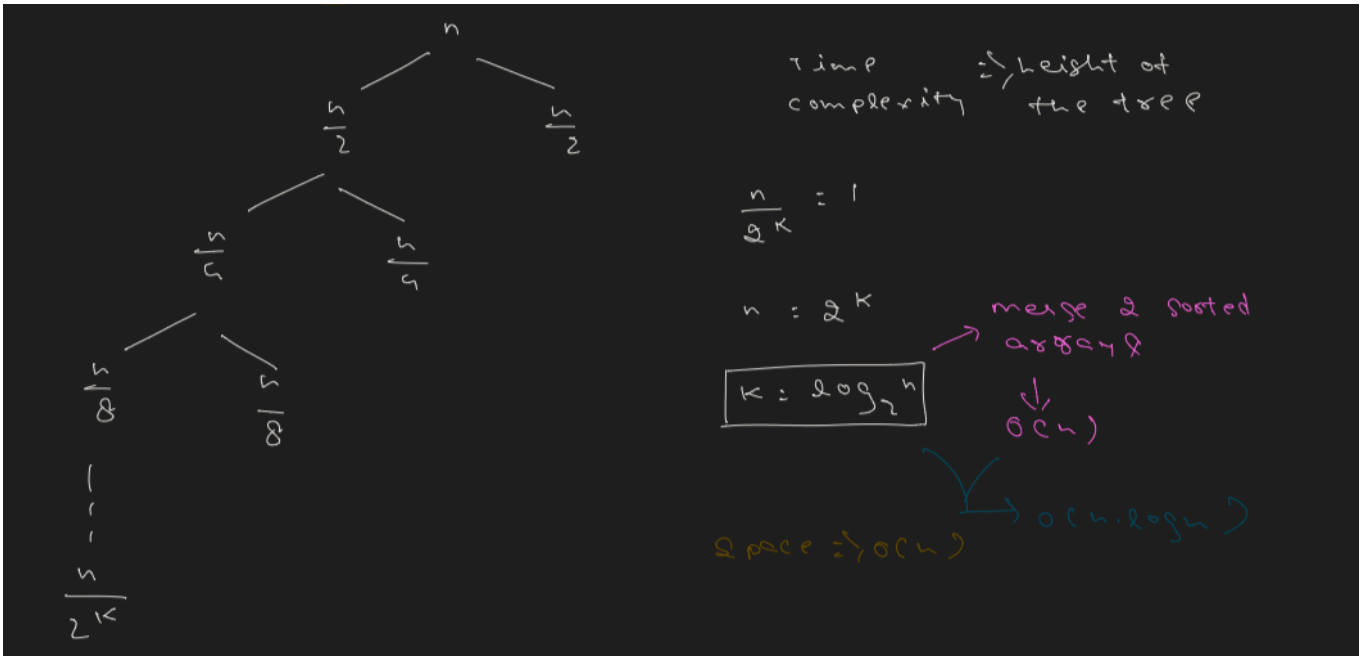
⚙️ How Merge Sort Works

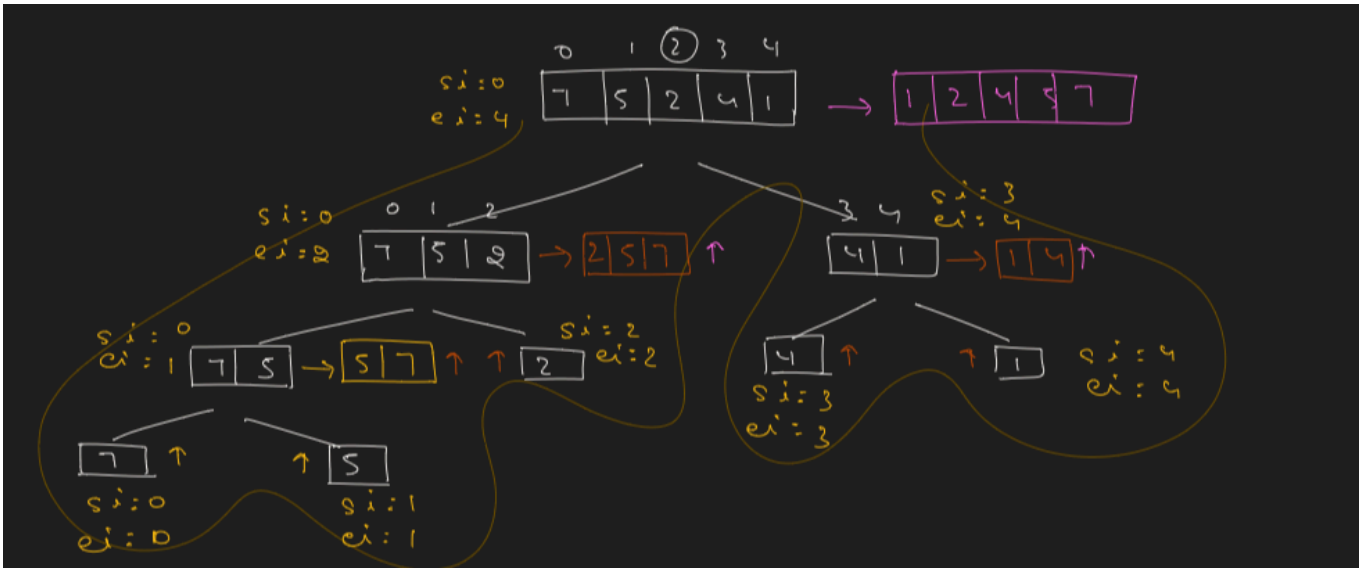
Merge Sort follows three main steps:

1. **Divide**:
   - Split the array into two halves.
   - Keep dividing each half until you're left with individual elements (size 1).
2. **Conquer**:
   - Sort each half recursively (though individual elements are already sorted).
3. **Combine (Merge)**:
   - Merge the two sorted halves into a single sorted array.










```java
public class Main {
    public static void main(String[] args) {
        int[] arr = { 7, 5, 2, 4, 1 };
        int[] a = sort(arr, 0, arr.length - 1);

        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i] + " ");
        }
    }

    public static int[] sort(int[] arr, int si, int ei) {
        if (si == ei) {
            int[] bs = { arr[si] };
            return bs;
        }

        int mid = (si + ei) / 2;
        int[] first = sort(arr, si, mid); // 1st part sorted by recursion
        int[] second = sort(arr, mid + 1, ei); // 2nd part sorted by recursion
        return merge(first, second);
    }
```

```java
    public static int[] merge(int[] arr1, int[] arr2) {
        int n = arr1.length;
        int m = arr2.length;
        int[] arr = new int[n + m];
        int i = 0;
        int j = 0;
        int k = 0;
        while (i < n && j < m) {
            if (arr1[i] < arr2[j]) {
                arr[k] = arr1[i];
                i++;
            } else {
                arr[k] = arr2[j];
                j++;
            }
            k++;
        }
        while (i < n) {
            arr[k] = arr1[i];
            k++;
            i++;
        }
        while (j < m) {
            arr[k] = arr2[j];
            k++;
            j++;
        }
        return arr;
    }

}
```

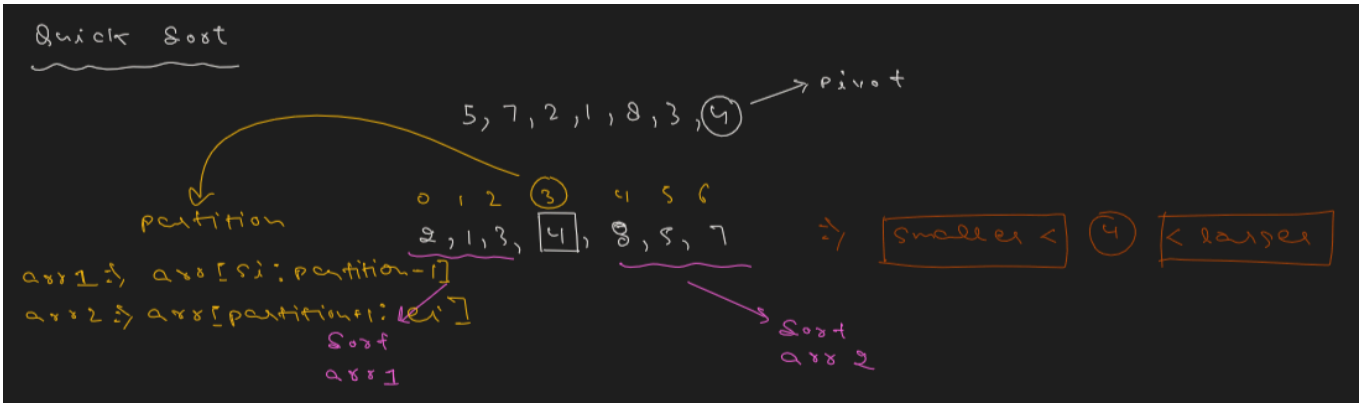
⚡ What is Quick Sort?

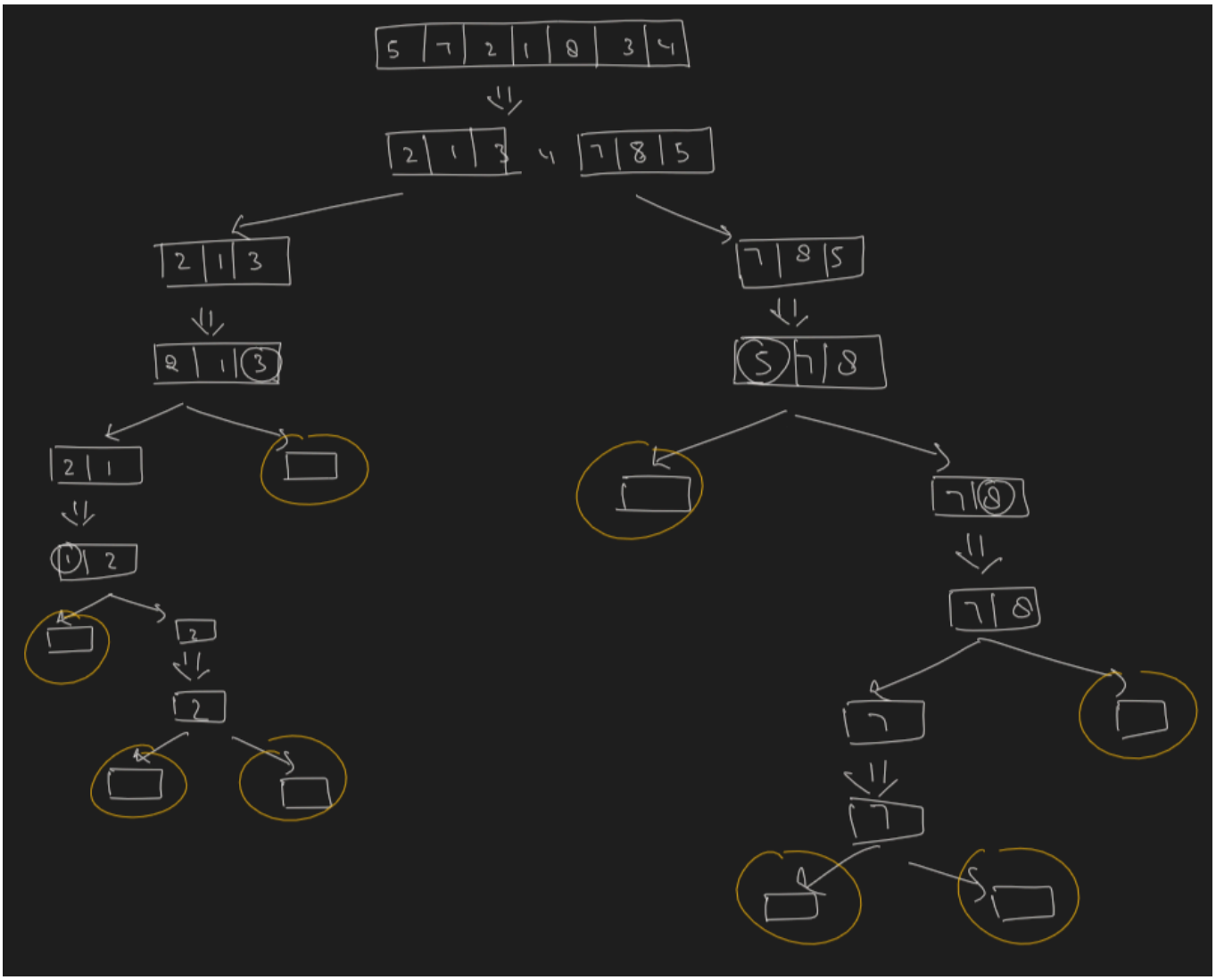
Quick Sort is a highly efficient **sorting algorithm** that uses the **Divide and Conquer** approach. It is one of the most commonly used sorting algorithms due to its average-case performance and in-place sorting nature (no extra space required).

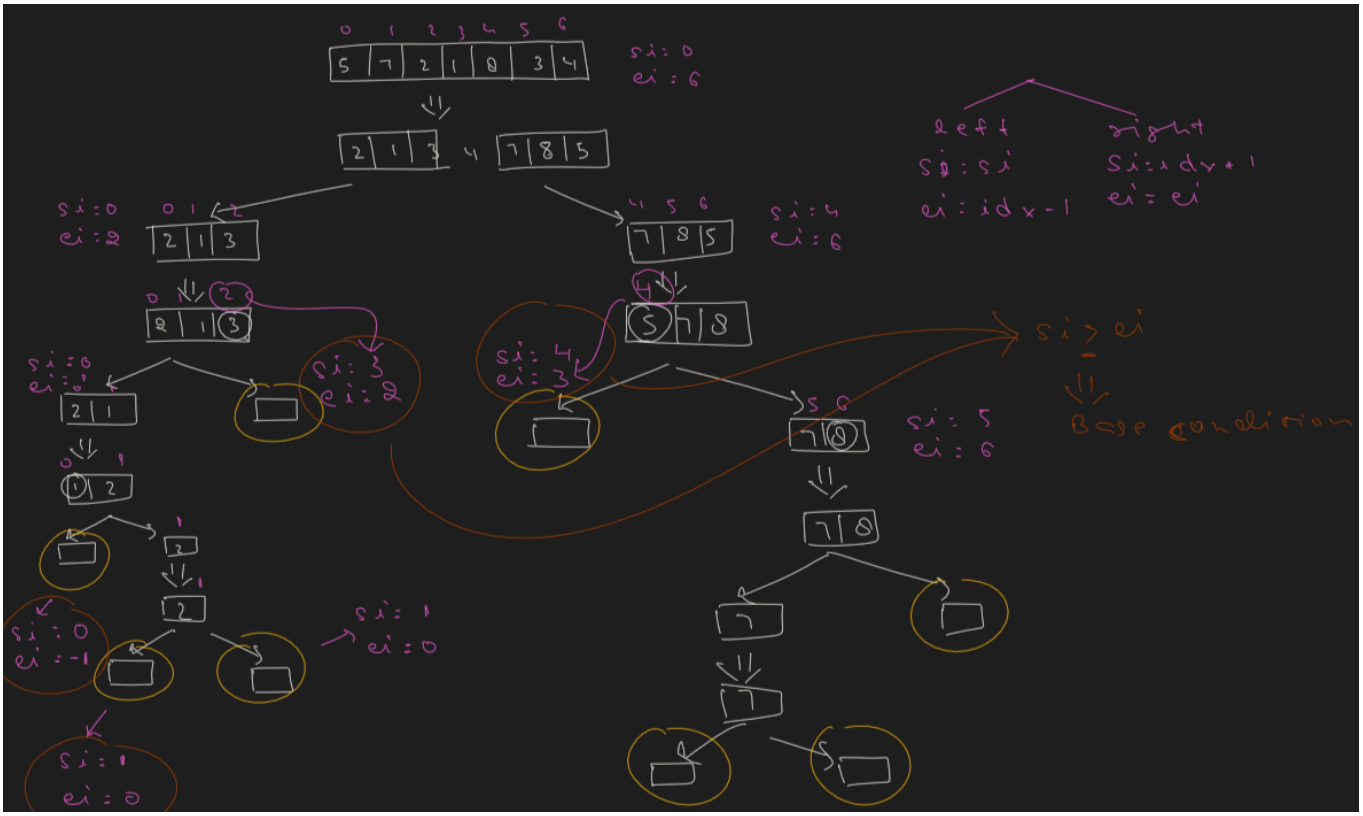
🧠 How Quick Sort Works

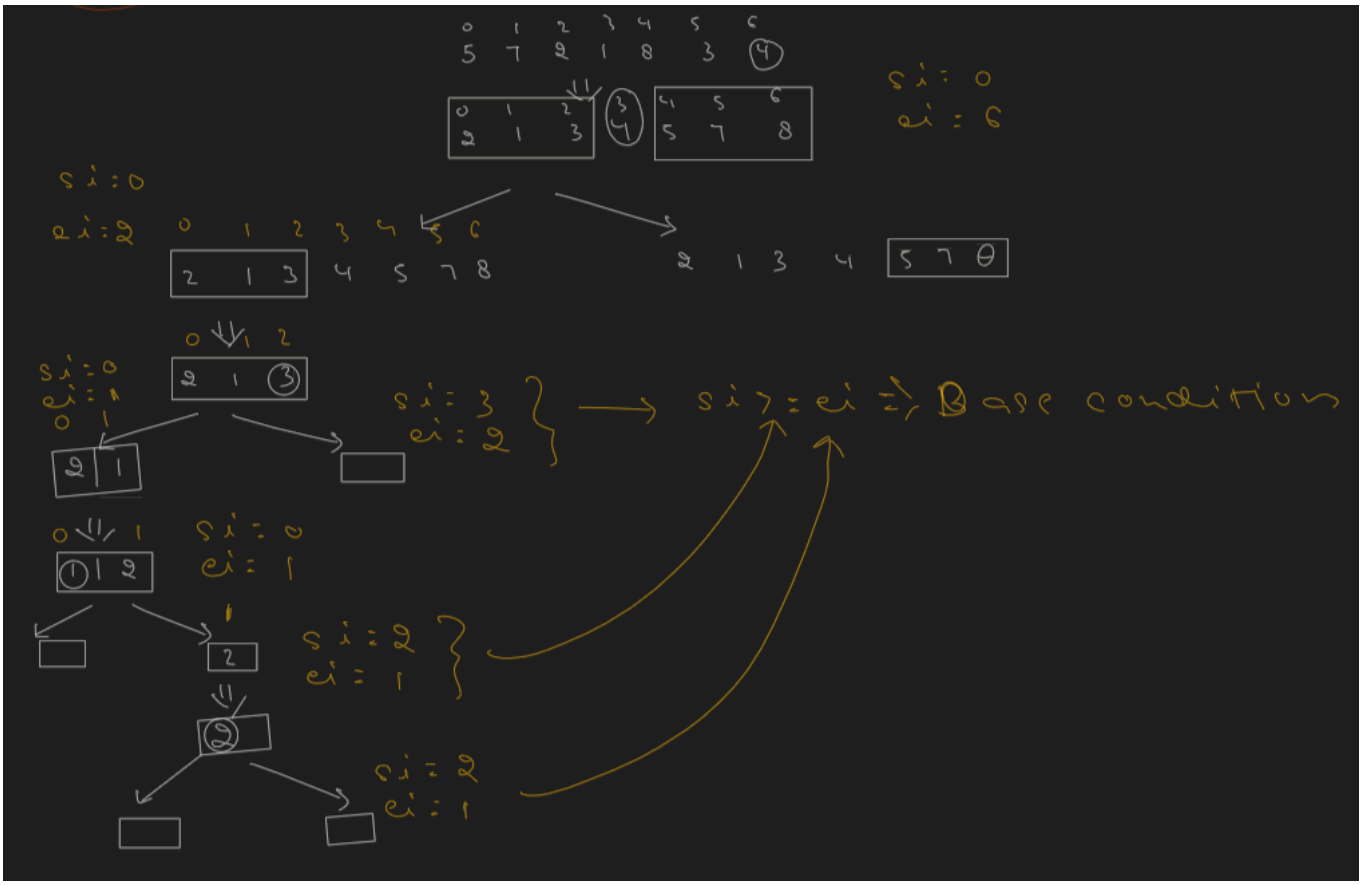
Quick Sort follows these steps:

1. **Pick a Pivot** – Choose an element from the array. This will be used to divide the array into parts.
2. **Partition** – Rearrange the array so that:
    - All elements **less than or equal** to the pivot come **before** it.
    - All elements **greater** than the pivot come **after** it.
3. **Recursively** apply Quick Sort to the **left and right** sub-arrays.
4. Base case: Stop when the sub-array has one or zero elements (already sorted).


Quick Sort
5, 7, 2, 1, 8, 3, 4
pivot
partition
0 1 2 3 4 5 6
2, 1, 3, 4, 8, 5, 7
=> smaller < 4 < larger
arr 1 => arr[s : partition-1]
arr 2 => arr[partition+1 : e]
Sort arr 1
Sort arr 2


5 7 2 1 8 3 4
2 1 3 4 7 8 5
2 1 3
7 8 5
2 1 3
5 7 8
2 1
1 2
2
2
7 8
7 8
7
7






```java
public class Main {
	public static void main(String[] args) {
		int[] arr = { 5, 7, 2, 1, 8, 3, 4 };
		sort(arr, 0, arr.length - 1);

		for (int i = 0; i < arr.length; i++) {
			System.out.print(arr[i] + " ");
		}
	}
```

```
      public static void sort(int[] arr, int si, int ei) {
            if (si >= ei) {
                  return;
            }
            int idx = partition(arr, si, ei);
            sort(arr, si, idx - 1);
            sort(arr, idx + 1, ei);
      }

      public static int partition(int[] arr, int si, int ei) {
            int item = arr[ei];
            int idx = si;
            for (int i = si; i < ei; i++) {
                  if (arr[i] <= item) {
                        int temp = arr[i];
                        arr[i] = arr[idx];
                        arr[idx] = temp;
                        idx++;
                  }

            }
            int temp = arr[ei];
            arr[ei] = arr[idx];
            arr[idx] = temp;
            return idx;
      }

}
```

🎲 Random numbers from a range

```
public class Main {

    public static void main(String[] args) {

            int lo = 10;
            int hi = 100;
            Random rn = new Random();

            System.out.println("y | x");
            System.out.println("------");

            for (int i = 0; i < 10; i++) {

                  // 91 --> 0 to 90 (will provide)
                  int y = rn.nextInt(hi - lo + 1);
                  System.out.print(y+"|");

                   // 0+10 to 90+10 (will provide)
                   int x = rn.nextInt(hi - lo + 1) + lo;
                   System.out.print(x+" ");
```

```
                System.out.println();
            }

        }

}
```